chapter 4

Input-Dependent and Asymptotic Approximation

Skip intro.

IN THE previous two chapters we have seen examples of NPO problems that can be approximated either within a specific constant factor or within any constant factor. We also saw examples of NPO problems for which no approximation algorithm exists (unless P=NP) and examples of NPO problems for which an approximation algorithm but no approximation scheme exists (unless P=NP). To deal with these two latter kinds of problem, in this chapter we will relax the constraint on the performance ratio in two ways.

We will first allow the performance ratio to be dependent on the input size. Clearly, any NPO problem for which a solution can be computed in polynomial time is approximable with respect to this weaker notion of performance ratio. Indeed, since the measure function can be computed in polynomial time, the ratio between the optimal measure and the measure of *any* feasible solution is always bounded by a function exponential in the length of the input. However, our goal is to find algorithms that produce solutions whose ratio is bounded by a more slowly increasing function. In particular, we will provide an $O(\log n)$-approximate algorithm for MINIMUM SET COVER, where n denotes the cardinality of the set to be covered, an $O(n/\log n)$-approximate algorithm for MINIMUM GRAPH COLORING, where n denotes the number of nodes of the graph, and an $O(\log k)$-approximate algorithm for MINIMUM MULTI-CUT, where k denotes the number of pairs of vertices that have to be disconnected.

We will then consider *asymptotic approximation schemes*, that is, schemes whose performance ratio is bounded by a constant, for any in-

Problem 4.1: Minimum Set Cover

INSTANCE: Collection C of subsets of a finite set S.

SOLUTION: A set cover for S, i.e., a subset $C' \subseteq C$ such that every element in S belongs to at least one member of C'.

MEASURE: $|C'|$.

stance whose optimal measure is large enough. In this case, we will show that MINIMUM EDGE COLORING and MINIMUM BIN PACKING admit an asymptotic approximation scheme. We have already seen in Sect. 2.2.2 that the latter problem belongs to the class APX and in Sect. 3.2.2 that it does not belong to PTAS (unless P=NP). We will see in this chapter that the same holds for the former problem. In a certain sense, the existence of an asymptotic approximation scheme shows that these two problems are easier than other problems in APX – PTAS.

4.1 Between APX and NPO

As we already saw in the preceding chapters, for several problems (such as MINIMUM TRAVELING SALESPERSON) it is possible to prove that a constant performance ratio is not achievable unless P = NP. In these cases we can relax the constant requirement on the ratio by looking for algorithms whose performance depends on the length of the instance. In this section we will give three examples of these algorithms.

4.1.1 Approximating the set cover problem

The first problem we consider is MINIMUM SET COVER (see Problem 4.1). For this problem, let us consider Program 4.1, which is a simple polynomial-time greedy procedure to cover set S. At each iteration, the algorithm chooses the set that covers the maximum number of uncovered elements (breaking ties arbitrarily) and updates the remaining sets.

Theorem 4.1 ▶ *Program 4.1 is a* $(\lfloor \ln n \rfloor + 1)$*-approximate algorithm for* MINIMUM SET COVER, *where* n *denotes the number of elements of the universe* S.

PROOF Given an instance of MINIMUM SET COVER, let k denote the largest cardinality of the sets in the collection. We will now prove that, for any optimal

Program 4.1: Greedy Set Cover

input Collection C of subsets of a finite set S;
output Set cover C';
begin
  $U := S$;
  for each set c_i in C **do** $c_i' := c_i$;
  $C' := \emptyset$;
  while $U \neq \emptyset$ **do**
  begin
    Let c_j' be a set with maximum cardinality;
    $C' := C' \cup \{c_j\}$;
    $U := U - c_j'$;
    for each c_i' **do** $c_i' := c_i' - c_j'$
  end;
  return C'
end.

Make a copy of collection C

Find the biggest set in the copy collection.
Put the original corresponding set in C'.
Disregard all its elements from S and from each set in the copy collection.
Repeat until all elements are disregarded.

solution C^*,

$$\sum_{c_i \in C^*} \mathcal{H}(|c_i|) \geq |C'|, \tag{4.1}$$

where, for any integer $r > 0$, $\mathcal{H}(r)$ denotes the r-th harmonic number (that is, $\mathcal{H}(r) = \sum_{i=1}^{r} 1/i$) and C' is the set cover obtained by Program 4.1. Since $|c_i| \leq k$, for any i with $1 \leq i \leq m$, $\mathcal{H}(k) \leq \lfloor \ln k \rfloor + 1$, and $k \leq n$, the above inequality implies that

$$|C'| \leq \sum_{c_i \in C^*} \mathcal{H}(k) \leq \mathcal{H}(n)|C^*| \leq (\lfloor \ln n \rfloor + 1)\,|C^*|.$$

That is, the performance ratio of Program 4.1 is at most $\lfloor \ln n \rfloor + 1$.

For any instance $x = (S, \{c_1, \ldots, c_m\})$ of MINIMUM SET COVER, let us denote by $a_1, \ldots, a_{|C'|}$ the sequence of indices of the subsets that belong to the set cover C'. Moreover, for any $j \in \{1, \ldots, |C'|\}$ and for any $i \in \{1, \ldots, m\}$, let c_i^j be the *surviving* part of c_i before index a_j has been chosen (that is, the subset of c_i that does not intersect any chosen set). Clearly, $c_i^1 = c_i$, for any i. Moreover, for the sake of simplicity, we will adopt the convention that $c_i^{|C'|+1} = \emptyset$, for any $i \in \{1, \ldots, m\}$. The set of elements of c_i that are covered for the first time by c_{a_j} is given by

$$c_i \cap c_{a_j}^j = c_i^j \cap c_{a_j}^j = c_i^j - c_i^{j+1}. \tag{4.2}$$

For any $i \in \{1, \ldots, m\}$, let l_i denote the largest index such that $|c_i^{l_i}| > 0$: that is, after $c_{a_{l_i}}$ has been included into C' the subset c_i has been covered.

The proof of Eq. (4.1) consists of the following two steps:

1. Prove that, for any $i \in \{1,\ldots,m\}$,
$$\mathcal{H}(|c_i|) \geq \sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|}.$$

2. Prove that, for any set cover C'' (and, therefore, for any optimal solution),
$$\sum_{c_i \in C''} \sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|} \geq |C'|.$$

Both steps consist of simple algebraic calculations. For what regards the first step, notice that, for any i with $1 \leq i \leq m$,

$$\begin{aligned}
\sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|} &= \sum_{j=1}^{|C'|} \frac{|c_i^j| - |c_i^{j+1}|}{|c_{a_j}^j|} \leq \sum_{j=1}^{l_i} \frac{|c_i^j| - |c_i^{j+1}|}{|c_i^j|} \\
&= \sum_{j=1}^{l_i} \sum_{k=|c_i^{j+1}|+1}^{|c_i^j|} \frac{1}{|c_i^j|} \leq \sum_{j=1}^{l_i} \sum_{k=1}^{|c_i^j|-|c_i^{j+1}|} \frac{1}{k+|c_i^{j+1}|} \\
&\leq \sum_{j=1}^{l_i} \left(\mathcal{H}(|c_i^j|) - \mathcal{H}(|c_i^{j+1}|)\right) = \mathcal{H}(|c_i^1|) = \mathcal{H}(|c_i|)
\end{aligned}$$

where the first equality follows from Eq. (4.2) and the first inequality is due to the fact that, for any j with $1 \leq j \leq |C'|$, $|c_i^j| \leq |c_{a_j}^j|$ (since the algorithm always chooses the biggest surviving set).

The second step follows as easily as the first one. Namely, for any set cover C'',

$$\sum_{c_i \in C''} \sum_{j=1}^{|C'|} \frac{|c_i \cap c_{a_j}^j|}{|c_{a_j}^j|} = \sum_{j=1}^{|C'|} \frac{1}{|c_{a_j}^j|} \sum_{c_i \in C''} |c_i \cap c_{a_j}^j| \geq \sum_{j=1}^{|C'|} \frac{|c_{a_j}^j|}{|c_{a_j}^j|} = |C'|$$

where the inequality is due to the fact that C'' is a set cover. In conclusion, Eq. (4.1) is satisfied and the theorem follows. QED

Example 4.1 ▶ Let us consider the instance of MINIMUM SET COVER with $S = \{1,\ldots,24\}$ and with C given by the following seven subsets of S: $c_1 = \{1,\ldots,8\}$, $c_2 = \{9,\ldots,16\}$, $c_3 = \{17,\ldots,24\}$, $c_4 = \{5,6,7,8,13,14,15,16,21,22,23,24\}$, $c_5 = \{3,4,11,12,19,20\}$, $c_6 = \{2,10,18\}$, $c_7 = \{1,9,17\}$. The first subset chosen by Program 4.1 is c_4. As a consequence, the first three subsets become $c_1^2 = \{1,\ldots,4\}$, $c_2^2 = \{9,\ldots,12\}$, and $c_3^2 = \{17,\ldots,20\}$, respectively. The next subset chosen is c_5 and, once again, the algorithm modifies the first three subsets that become $c_1^3 = \{1,2\}$, $c_2^3 = \{9,10\}$, and $c_3^3 = \{17,18\}$, respectively. At the next two iterations of the loop, the subsets chosen are c_6 and c_7, respectively, so that the solution computed by the algorithm uses four subsets. On the other hand, it is easy to verify that the first three subsets form an optimal solution.

Program 4.2: Greedy Graph Coloring

```
input Graph G = (V,E);
output Node coloring of G;
begin
  i := 0;
  U := V;
  while U ≠ ∅ do
  begin
    i := i + 1;
    W := U;
    while W ≠ ∅ do
    begin
      H := graph induced by W;
      v := node of minimum degree in H;
      f(v) := i;
      W := W − {v} − {z|z is a neighbor of v in H}
    end;
    U := U − f^{-1}(i);
  end;
  return f
end.
```

The analysis of Theorem 4.1 is tight: indeed, it is possible to generalize the previous example in order to prove such statement (see Exercise 4.1).

4.1.2 Approximating the graph coloring problem

In this section we provide an $O(n/\log n)$-approximate algorithm for MINIMUM GRAPH COLORING based on repeatedly finding an independent set (i.e., a set of vertices that can be colored with one color) and coloring it with a new color, until all vertices have been colored. This algorithm is given as Program 4.2 and, clearly, runs in polynomial time. The next result gives an upper bound on its performance ratio.

◀ Lemma 4.2

Program 4.2 colors any k-colorable graph $G = (V,E)$ with at most $3|V|/\log_k |V|$ colors.

PROOF

Let H be the graph considered by the algorithm at the beginning of a generic iteration of the inner loop and let W be the corresponding set of vertices. Since H is a subgraph of a k-colorable graph, H is a k-colorable graph and, therefore, it must contain an independent set of size at least $|W|/k$. Each vertex of this set has degree at most $|W| - |W|/k =$

$|W|(k-1)/k$. This implies that the minimum degree of H is at most $|W|(k-1)/k$ so that at least $|W| - |W|(k-1)/k = |W|/k$ nodes will still be in W at the beginning of the next iteration. Since the inner loop ends when W becomes empty, we have that at least $\lceil \log_k |W| \rceil$ iterations have to be performed, which in turn implies that, at the end of the inner loop,

$$|\{v \mid v \in W \wedge f(v) = i\}| \geq \lceil \log_k |W| \rceil.$$

Hence, for each used color i, the number of vertices colored with i is at least $\lceil \log_k |U| \rceil$, where U denotes the set of nodes uncolored just before color i is considered.

Let us now analyze the size of U at the beginning of an iteration of the outer loop. If $|U| \geq |V|/\log_k |V|$, then we have that

$$\lceil \log_k |U| \rceil \geq \log_k |U| \geq \log_k (|V|/\log_k |V|) > \log_k \sqrt{|V|} = \frac{1}{2} \log_k |V|.$$

This implies that the size of U decreases by at least $\frac{1}{2} \log_k |V|$ at each iteration: as a consequence, the first time $|U|$ becomes smaller than $|V|/\log_k |V|$, the algorithm has used no more than $2|V|/\log_k |V|$ colors.

If $|U| < |V|/\log_k |V|$, it is clear that the algorithm colors all remaining nodes in U with at most $|U| < |V|/\log_k |V|$ colors. It follows that the algorithm uses at most $3|V|/\log_k |V|$ colors.

QED

Theorem 4.3 ▶ MINIMUM GRAPH COLORING *admits an $O(n/\log n)$-approximate algorithm, where n denotes the number of nodes.*

PROOF The previous lemma implies that, for any input graph G with n nodes, the solution returned by Program 4.2 uses at most $3n/\log_{m^*(G)} n = 3n \log(m^*(G))/\log n$ colors, where $m^*(G)$ denotes the minimum number of colors necessary to color G. This implies that the performance ratio of this algorithm is at most

$$\frac{3n \log(m^*(G))/\log n}{m^*(G)} = O(n/\log n)$$

QED and the theorem follows.

Example 4.2 ▶ Let us consider the graph of Fig. 4.1(a). In this case $m^*(G) = 3$: indeed, we can assign to nodes a, d, and h the first color, to nodes b, e, and g the second color, and to nodes c and f the third color. Program 4.2, instead, could first choose node a and assign to it the first color. The resulting graph H is shown in Fig. 4.1(b): at this step, node e is chosen and the first color is assigned to it. The new graph H is shown in Fig. 4.1(c): in this case, the algorithm could choose node h and assign to it the first color. In conclusion, after exiting the inner loop, nodes a, e, and h have been assigned the first color.

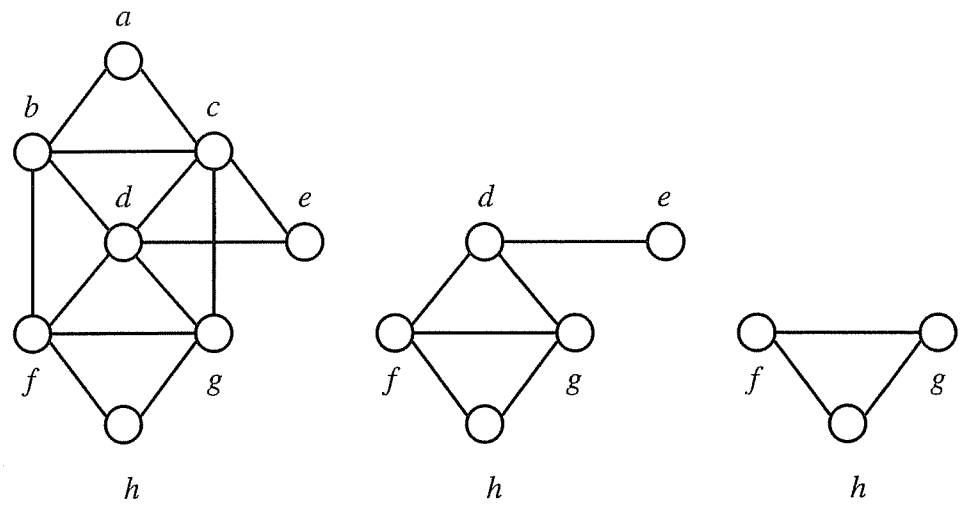

Figure 4.1
The first iteration of the outer loop of Program 4.2

Since uncolored nodes exist (see Fig. 4.2(a)), the algorithm will continue executing the outer loop and a possible run of the algorithm could assign the second color to nodes b and g. Successively, the third color could be assigned to nodes c and f and, finally, the fourth color is assigned to node d (see Figs. 4.2(a) and (b)). Hence, in this case the algorithm produces a 4-coloring.

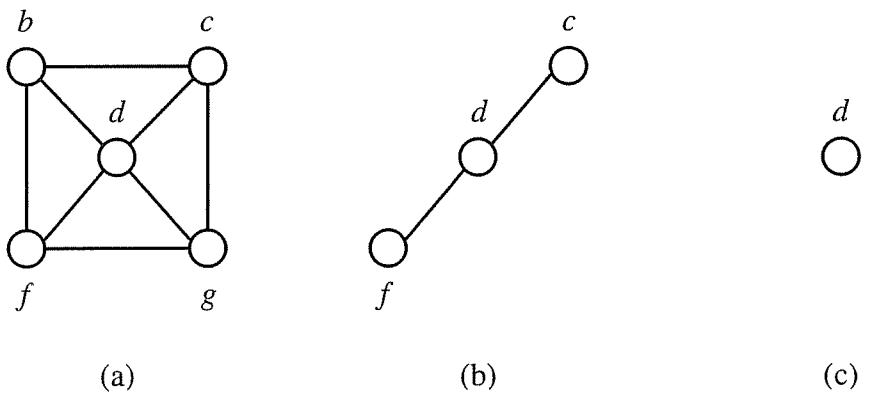

Figure 4.2
The next steps of Program 4.2

4.1.3 Approximating the minimum multi-cut problem

In this section, we give another example of how duality relationships between integer linear programming problems (see Appendix A) can be exploited to derive efficient approximation algorithms (a first example of an approximation algorithm based on the duality relationship has been given in Sect. 2.4.2).

First of all, let us recall that MAXIMUM FLOW (see Problem 4.2) consists in finding the maximum flow that can be routed in a network from a

Problem 4.2: Maximum Flow

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, pair $(s,t) \in V \times V$.

SOLUTION: A flow from s to t, that is, a function $f : V \times V \mapsto \mathbb{Q}^+$ such that:

1. $f(u,v)$ is defined if and only if $(u,v) \in E$;
2. $f(u,v) + f(v,u) \leq c(u,v)$, for each $(u,v) \in E$, that is, the total flow in an edge cannot exceed its capacity;
3. $\sum_{u \in V-\{v\}} f(u,v) = \sum_{u \in V-\{v\}} f(v,u)$ for each $v \in V - \{s,t\}$, that is, the flow entering in a node must equate the flow exiting the same node, for all nodes except the source and the destination.

MEASURE: $\sum_{u \in V-\{t\}} f(u,t) - \sum_{u \in V-\{t\}} f(t,u)$, that is, the total flow entering the destination.

source s to a destination t without violating some specified edge capacity constraints. Moreover, MINIMUM CUT (see Problem 4.3) requires to find the set of arcs of minimum total capacity whose removal disconnects the source from the destination in a network. These two problems are related through the maxflow-mincut theorem (see Bibliographical notes), a central result in flow theory, that lies at the basis of the solution of many relevant optimization problems. Such theorem states that, since a duality relationship holds between MAXIMUM FLOW and MINIMUM CUT, their optimal solutions have the same value.

Let us now consider the generalization of MAXIMUM FLOW to the case in which more than one commodity has to flow in the network (see Problem 4.4). This problem has the following linear programming formulation LP_{MMF}:

$$\begin{aligned}
\texttt{maximize} \quad & \sum_{i=1}^{k} \sum_{v \in V-\{t_i\}} (f^i_{v,t_i} - f^i_{t_i,v}) & \\
\texttt{subject to} \quad & \sum_{(v,u) \in E} f^i_{vu} - \sum_{(u,v) \in E} f^i_{uv} = 0 & \forall v \in V - \{s_i,t_i\},\ i = 1,\ldots,k \\
& \sum_{i=1}^{k} f^i_{uv} + \sum_{i=1}^{k} f^i_{vu} \leq c_{uv} & \forall (u,v) \in E \\
& f^i_{uv} \geq 0 & \forall (u,v) \in E,
\end{aligned}$$

where f^i_{uv} is the flow of commodity i along edge (u,v) from u to v) and c_{uv} denotes the capacity of edge (u,v). The first set of constraints represents

Problem 4.3: Minimum Cut

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, pair $(s,t) \in V \times V$.

SOLUTION: A set $E' \subseteq E$ of edges whose removal disconnects s and t.

MEASURE: The overall capacity of E', i.e., $\sum_{e \in E'} c(e)$.

flow conservation at vertices, while the second set of inequalities states that the total flow value assigned to each edge does not exceed its capacity.

Let us also consider the corresponding generalization of MINIMUM CUT, shown in Problem 4.5. This problem has the following integer linear programming formulation:

$$\begin{array}{lll} \texttt{minimize} & \sum_{(u,v)\in E} d_{uv}c_{uv} & \\ \texttt{subject to} & \sum_{(u,v)\in E} d_{uv}\pi_i^j(u,v) \geq 1 & \forall \pi_i^j \in \mathcal{P}_i, i = 1,\ldots,k \\ & d_{uv} \in \mathbb{N} & \forall (u,v) \in E, \end{array}$$

where $\mathcal{P}_i$ denotes the set of all paths connecting s_i to t_i, π_i^j is the j-th of such paths, $\pi_i^j(u,v)$ is 1 if (u,v) is an edge in π_i^j, and $\pi_i^j(u,v)$ is 0 otherwise. Notice that, indeed, any solution of the above problem corresponds to a multi-cut, defined as the set of edges (u,v) such that $d_{uv} > 0$. Notice also that the above formulation has an exponential number of constraints: however, an equivalent, less natural, formulation ILP_{MC} with polynomially many constraints is the following one (see Exercise 4.8):

$$\begin{array}{lll} \texttt{minimize} & \sum_{(u,v)\in E} d_{uv}c_{uv} & \\ \texttt{subject to} & d_{uv} \geq p_u^i - p_v^i & \forall (u,v) \in E,\ i = 1,\ldots,k \\ & p_s^i - p_t^i \geq 1 & i = 1,\ldots,k \\ & p_v^i \in \mathbb{N} & \forall v \in V,\ i = 1,\ldots,k \\ & d_{uv} \in \mathbb{N} & \forall (u,v) \in E, \end{array}$$

where d_{uv} is a variable that, by the minimization requirement, will assume value at most 1 in the optimal solution and with such a value will denote that edge (u,v) is in the multi-cut. Intuitively, variables p_v^i denote a *potential* associated to flow i at vertex v. In this way, the constraints impose that, for each pair (s_i,t_i), they must be sufficiently "far apart" to always have at least one edge in the multi-cut between them.

It is possible to see that, by applying the well-known primal-dual transformation (see Appendix A) to LP_{MMF}, we obtain the same linear program

Problem 4.4: Maximum Multi-commodity Flow

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, k pairs $(s_i,t_i) \in V \times V$.

SOLUTION: A set of k flows f^i, each from s_i to t_i, for $i = 1,\ldots,k$, such that the total flow value $\sum_{\forall i} f^i(u,v)$ that passes through edge (u,v) does not exceed its capacity.

MEASURE: The sum of the total flows entering all destinations.

Problem 4.5: Minimum Multi-Cut

INSTANCE: Graph $G = (V,E)$, edge capacity function $c : E \mapsto \mathbb{Q}^+$, k pairs $(s_i,t_i) \in V \times V$.

SOLUTION: A subset $E' \subseteq E$ of edges whose removal disconnects s_i and t_i, $i = 1,\ldots,k$.

MEASURE: The overall capacity $\sum_{e \in E'} c(e)$ of E'.

as by relaxing the integrality constraints of ILP_{MC}. Such a program is the linear programming formulation of MINIMUM FRACTIONAL MULTI-CUT. This situation is similar to what happens in the case of MAXIMUM FLOW, whose dual is MINIMUM FRACTIONAL CUT, i.e., the relaxation of MINIMUM CUT to include also non integral solutions. However, the combinatorial structure of MINIMUM FRACTIONAL CUT guarantees that any optimal solution is an integral one and, as a consequence, an optimal solution also for MINIMUM CUT. Unfortunately, the equality does not hold for MINIMUM FRACTIONAL MULTI-CUT and all we can say in this case is that the capacity of the minimum multi-cut in a graph is *at least* the value of the maximum multi-commodity flow in the same graph.

Actually, a stronger relationship has been shown between MINIMUM MULTI-CUT and MAXIMUM MULTI-COMMODITY FLOW: given a graph $G = (V,E)$ with k commodities, the capacity of the minimum multi-cut is at most $O(\log k)$ times the maximum flow in G.

In the following, we present an approximation algorithm that, given an instance of MINIMUM MULTI-CUT, returns a solution whose capacity is bounded by $O(\log k)$ times the maximum multi-commodity flow in the graph. This guarantees that the value of such a solution is at most $O(\log k)$ times the capacity of the minimum multi-cut. The algorithm exploits the knowledge of an optimal fractional multi-cut, which can be efficiently computed by solving in polynomial time either MINIMUM FRACTIONAL

MULTI-CUT or its dual MAXIMUM MULTI-COMMODITY FLOW.

Let $d : E \mapsto \mathbb{Q}^+$ be a function that associates to each edge (u,v) the value of variable d_{uv} in the optimal solution of MINIMUM FRACTIONAL MULTI-CUT. If we consider the graph $G = (V,E)$ with functions $c : E \mapsto \mathbb{Q}^+$ and $d : E \mapsto \mathbb{Q}^+$ and k pairs $\{(s_1,t_1),\ldots,(s_k,t_k)\}$, we may see it as a set of pipes (edges) connected at nodes, where pipe (u,v) has length $d(u,v)$ and cross section $c(u,v)$ (and thus volume $c(u,v)d(u,v)$). In particular, this pipe system is the one of minimum volume such that, for each pair (s_i,t_i), the distance between s_i and t_i is at least 1. Moreover, let $\Psi = \sum_{(u,v)\in E} c(u,v)d(u,v)$ be the overall volume of G: clearly, since MINIMUM FRACTIONAL MULTI-CUT is a relaxation of MINIMUM MULTI-CUT, Ψ is a lower bound on the optimal measure of the corresponding MINIMUM MULTI-CUT problem.

The algorithm (see Program 4.3) works by iteratively producing a sequence $\mathcal{V} = (V_1, V_2, \ldots, V_q)$ of disjoint subsets of V such that, for each $r = 1,\ldots,q$:

1. at most one between s_i and t_i is contained in V_r, for each $i = 1,\ldots,k$;
2. there exists $l \in \{1,\ldots,k\}$ such that either s_l or t_l is contained in V_r.

Note that, for any r with $1 \leq r \leq q$, the subsequence $(V_1,\ldots,V_r)$ *separates* at least r pairs (s_i,t_i).

The algorithm works in phases, where at each phase a new set V_r is added to $\mathcal{V}$, until each vertex is included in one subset of $\mathcal{V}$. In particular, let $\overline{G}_r = (\overline{V}_r, \overline{E}_r)$ be the graph induced by $\overline{V}_r = V - (\cup_{j=1}^{r-1} V_j)$: if there exists no pair (s_i,t_i) such that $s_i \in \overline{V}_r$ and $t_i \in \overline{V}_r$, then the algorithm sets $V_r = \overline{V}_r$ and returns the multi-cut $\{(u,v) \in E \mid u \in V_p \wedge v \in V_q \wedge p \neq q\}$.

Otherwise, let (s_i,t_i) be an arbitrarily chosen pair such that both $s_i \in \overline{V}_r$ and $t_i \in \overline{V}_r$. The algorithm then computes a suitable set $V_r \subseteq \overline{V}_r$ such that exactly one between s_i and t_i is contained in V_r. This is performed by selecting a set of vertices that are within a certain distance ρ from s_i. This set of vertices is a *ball* centered at s_i and of radius ρ. The algorithm starts with $\rho = 0$ and iteratively increases ρ until a suitable condition is verified.

In order to describe more in detail how this is done, let us first introduce some definitions. For any $u,v \in V$, let $\delta(u,v)$ be the length of a shortest path from u to v, with respect to edge length $d : E \mapsto \mathbb{Q}^+$. For any $v \in \overline{V}_r$ and for any $\rho \in \mathbb{Q}$, the *ball* of radius ρ around v (with respect to δ) is defined as

$$\mathcal{B}_\delta(v,\rho) = \{u \in \overline{V}_r \mid \delta(u,v) \leq \rho\}.$$

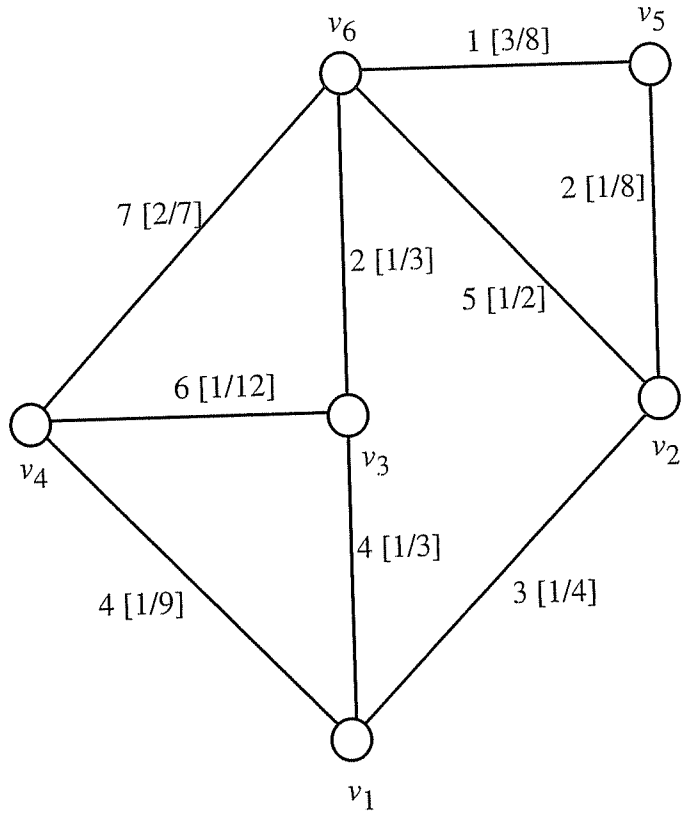

Figure 4.3
Sample graph G: square brackets contain the values of function d

Moreover, let

$$\mathcal{E}_\delta(v,\rho) = \{(w,z) \in \overline{E}_r \mid w \in B_\delta(v,\rho) \wedge z \in B_\delta(v,\rho)\}$$

be the set of edges completely contained in $\mathcal{B}_\delta(v,\rho)$ and let

$$\overline{\mathcal{E}}_\delta(v,\rho) = \{(w,z) \in \overline{E}_r - \mathcal{E}_\delta(v,\rho) \mid w \in B_\delta(v,\rho) \vee z \in B_\delta(v,\rho)\}$$

be the set of edges partially contained in $\mathcal{B}_\delta(v,\rho)$. Finally, the *volume* of $\mathcal{B}_\delta(v,\rho)$ is defined as

$$V_\delta(v,\rho) = \frac{\Psi}{k} + \sum_{(w,z)\in\mathcal{E}_\delta(v,\rho)} c(w,z)d(w,z) + \sum_{(w,z)\in\overline{\mathcal{E}}_\delta(v,\rho)} c(w,z)(\rho - \min(\delta(v,w),\delta(v,z))).$$

while the *cost* of $\mathcal{B}_\delta(v,\rho)$ is defined as

$$C_\delta(v,\rho) = \sum_{(w,z)\in\overline{E}_\delta(v,\rho)} c(w,z).$$

Note that, apart from a fixed amount of volume Ψ/k (that we assume located on v), each edge contributes to the volume of $\mathcal{B}_\delta(v,\rho)$ for its fraction contained in $\mathcal{B}_\delta(v,\rho)$. On the other side, an edge contributes to the cost of $\mathcal{B}_\delta(v,\rho)$ for its capacity, if it is in the cut separating $\mathcal{B}_\delta(v,\rho)$ from $V - \mathcal{B}_\delta(v,\rho)$.

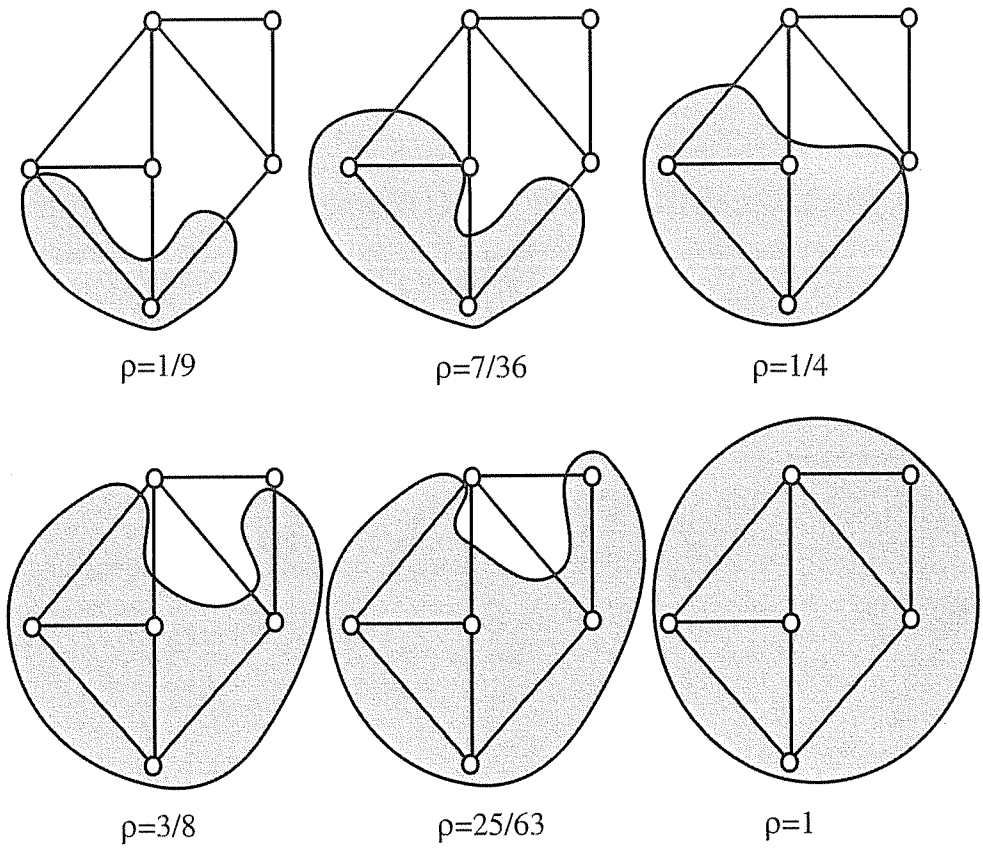

Figure 4.4
The balls $B_\delta(v_1,\rho)$ for som values of ρ

◀ Example 4.3

Let us consider the graph shown in Fig. 4.3, where the square brackets contain the values of function d, and assume that $k=2$. Hence, the amount of volume that the algorithm assumes to be located at each node is

$$\frac{\Psi}{k}=\frac{213/24}{2}=\frac{213}{48}\approx 4.4.$$

Moreover, if we consider v_1 as the center of the balls, we have that $\delta(v_1,v_2)=1/4$, $\delta(v_1,v_3)=7/36$, $\delta(v_1,v_4)=1/9$, $\delta(v_1,v_5)=3/8$, $\delta(v_1,v_6)=25/63$. In Fig. 4.4 we show the balls centered in v_1 corresponding to $\rho=\delta(v_1,v_4)-\varepsilon$, $\rho=\delta(v_1,v_3)-\varepsilon$, $\rho=\delta(v_1,v_2)-\varepsilon$, $\rho=\delta(v_1,v_5)-\varepsilon$, $\rho=\delta(v_1,v_6)-\varepsilon$, and $\rho=1-\varepsilon$, where ε is abitrarily small. Let us now consider, for example, $\mathcal{B}_\delta(v_1,1/4-\varepsilon)$. This ball contains vertices v_1, v_3, and v_4 and we have that $\mathcal{E}_\delta(v_1,1/4-\varepsilon)=\{(v_1,v_3),(v_1,v_4),(v_3,v_4)\}$ and that $\overline{\mathcal{E}}_\delta(v_1,1/4-\varepsilon)=\{(v_1,v_2),(v_3,v_6),(v_4,v_6)\}$. Hence, it results

$$\begin{aligned}V_\delta(v_1,\frac{1}{4}-\varepsilon) &= \frac{\Psi}{k}+\frac{4}{3}+\frac{4}{9}+\frac{1}{2}+3(\frac{1}{4}-\varepsilon)+2(\frac{1}{4}-\varepsilon-\frac{7}{36})+7(\frac{1}{4}-\varepsilon-\frac{1}{9})\\ &= \frac{213}{48}+16.8+\frac{10}{9}-3\varepsilon\approx 22.31\end{aligned}$$

and

$$C_\delta(v_1,1/4-\varepsilon)=3+2+7=12.$$

Observe that we can consider the ratio $C_\delta(v,\rho)/V_\delta(v,\rho)$ as a function of ρ. Let $\{u_1, u_2, \ldots, u_{|\overline{V}_r|-1}\}$ be the set of nodes in $\overline{V}_r - \{v\}$ ordered by non decreasing distance from v (that is, $i < j \Rightarrow \delta(v,u_i) \leq \delta(v,u_j)$) and let $\{r_1, r_2, \ldots, r_{|\overline{V}_r|-1}\}$ be the corresponding distances from v. It is easy to see that the function $C_\delta(v,\rho)/V_\delta(v,\rho)$ is continuous and non-increasing, for $0 < \rho < r_1$ and for $r_i < \rho < r_{i+1}$ $(i = 1, \ldots, |\overline{V}_r| - 2)$. This implies that it tends to its minimum as it approaches some r_j from below. Notice, moreover, that the derivative of $C_\delta(v,\rho)/V_\delta(v,\rho)$ with respect to ρ is well defined for all values $\rho \notin \{r_1, \ldots, r_{|\overline{V}_r|-1}\}$ (see Fig. 4.3).

Example 4.4 ▶ Let us consider again the graph shown in Fig. 4.3 and assume that v_1 is the center of the balls. In this case, $u_1 = v_4$, $u_2 = v_3$, $u_3 = v_2$, $u_4 = v_5$, and $u_5 = v_6$. Moreover, $r_1 = 1/9$, $r_2 = 7/36$, $r_3 = 1/4$, $r_4 = 3/8$, and $r_5 = 25/63$. In Fig. 4.5 the behavior of function $C_\delta(v,\rho)/V_\delta(v,\rho)$ is shown in the interval of interest (that is, $(0, 25/63)$).

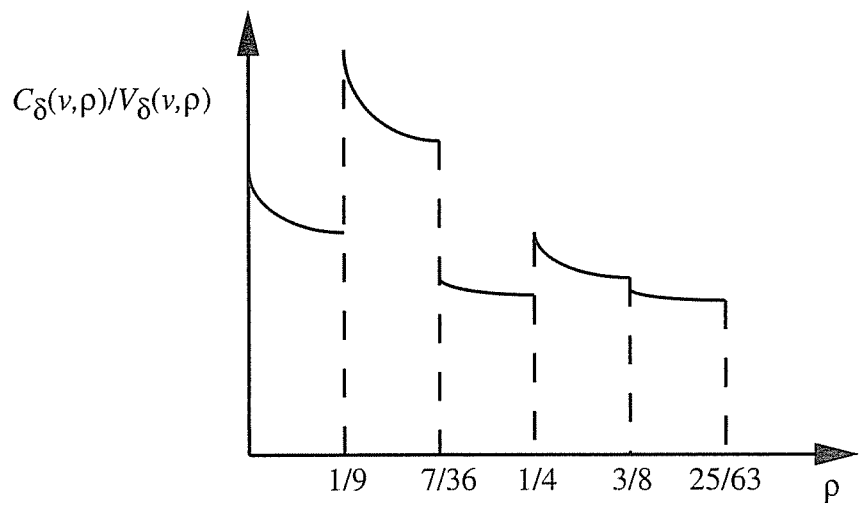

Figure 4.5
The function $C_\delta(v,\rho)/V_\delta(v,\rho)$

Let us now prove that there exists a value $\rho_r < 1/2$ for which the corresponding value $C_\delta(v,\rho_r)/V_\delta(v,\rho_r)$ is sufficiently small.

Lemma 4.4 ▶ *There exists a value* $\rho_r < 1/2$ *such that*

$$\frac{C_\delta(v,\rho_r)}{V_\delta(v,\rho_r)} \leq 2\ln 2k.$$

PROOF Let us first notice that $C_\delta(v,\rho)$ is the derivative of $V_\delta(v,\rho)$ with respect to ρ. As a consequence, whenever such a derivative is defined, we have

$$\frac{C_\delta(v,\rho)}{V_\delta(v,\rho)} = \frac{\partial}{\partial\rho}(\ln V_\delta(v,\rho)).$$

Assume now, by contradiction, that, for every $\rho \in (0,1/2)$, we have $C_\delta(v,\rho)/V_\delta(v,\rho) > 2\ln 2k$. Two cases are possible:

1. $r_1 \geq 1/2$. This implies that $C_\delta(v,\rho)/V_\delta(v,\rho)$ is differentiable in $(0,1/2)$. As a consequence, we have that

$$\frac{\partial}{\partial\rho}(\ln V_\delta(v,\rho)) > 2\ln 2k$$

and, by integrating both sides in $(0,1/2)$,

$$\ln\frac{V_\delta(v,1/2)}{V_\delta(v,0)} > \ln 2k,$$

which implies $V_\delta(v,1/2) > 2kV_\delta(v,0) = 2\Psi$. This contradicts the fact that $\Psi + \Psi/k$ is the maximum possible value of $V_\delta(v,\rho)$, which is obtained when $\rho \geq r_{|\overline{V}_r|-1}$, i.e., when all the graph is included.

2. There exists t, with $1 \leq t \leq |\overline{V}_r| - 1$, such that $r_1,\ldots,r_t \in (0,1/2)$ (therefore, $C_\delta(v,\rho)/V_\delta(v,\rho)$ might not be differentiable in $(0,1/2)$). Let $r_0 = 0$ and $r_{t+1} = 1/2$. If we apply the same considerations above to each interval (r_i, r_{i+1}), for $i = 0,\ldots,t$, we obtain that

$$\ln V_\delta(v,r_{i+1}) - \ln V_\delta(v,r_i) > (r_{i+1} - r_i)2\ln 2k.$$

Since the volume does not decrease in correspondence to each vertex, by summing over all intervals we get

$$\ln\frac{V_\delta(v,1/2)}{V_\delta(v,0)} = \ln V_\delta(v,1/2) - \ln V_\delta(v,0) > \ln 2k,$$

thus reducing to the previous case.

The lemma is thus proved. QED

We are now ready to prove the main result of this section.

◀ Theorem 4.5 *Given an instance x of* MINIMUM MULTI-CUT, *the solution returned by Program 4.3 with input x is a feasible solution whose cost $m_{MC}(x)$ satisfies the inequality*

$$m_{MC}(x) < 4\ln(2k)m^*(x).$$

PROOF Let us first observe that the solution returned by the algorithm is feasible. Indeed, since distances $d(u,v)$ are derived by solving MINIMUM FRACTIONAL MULTI-CUT, we have that $d(s_i,t_i) \geq 1$, for $i = 1,\ldots,k$. As we may observe, for all $i = 1,\ldots,k$, Program 4.3 separates s_i from all vertices at distance at least 1/2: in particular, it separates s_i from t_i. Hence, the solution returned by the algorithm is a multi-cut.

Program 4.3: Multi-cut

```
input Graph G = (V,E), pairs (s_i,t_i), functions c,d;
output Multi-cut E';
begin
  r := 1;
  V_r := V;
  while V_r ≠ ∅ do begin
    if there exists i such that s_i ∈ V_r ∧ t_i ∈ V_r then
    begin
      Let ρ_r be such that C_δ(s_i,ρ_r)/V_δ(s_i,ρ_r) ≤ C_δ(s_i,ρ)/V_δ(s_i,ρ)
        for all ρ = δ(s_i,u) − ε, u ∈ V and ρ < 1/2;
      V_r := B_δ(s_i,ρ_r);
      r := r+1
    V_r := V_{r−1} − V_{r−1};
    end
    else
      V_r := ∅
  end;
  E' := set of edges between vertices in V_i and vertices in V_j, for i ≠ j;
  return E'
end.
```

For what concerns the performance ratio, let us now prove that $m_{MC}(x) < \ln(2k)\Psi$. The theorem will then follow since Ψ is a lower bound on the optimal measure of MINIMUM MULTI-CUT.

Let $V_1, V_2, \ldots, V_h$ ($h \leq k$) be the subsets of V produced by Program 4.3. Let $V_\delta(s_{i_1}, \rho_1), V_\delta(s_{i_2}, \rho_2), \ldots, V_\delta(s_{i_h}, \rho_h)$ be the volumes of the corresponding balls and let $C_\delta(s_{i_1}, \rho_1), C_\delta(s_{i_2}, \rho_2), \ldots, C_\delta(s_{i_h}, \rho_h)$ be the costs of the corresponding cuts.

By Lemma 4.4, the cost of the multi-cut returned by the algorithm is

$$m_{MC}(x) = \sum_{j=1}^{h} C_\delta(s_{i_j}, \rho_j) \leq 2\ln(2k) \sum_{j=1}^{h} V_\delta(s_{i_j}, \rho_j).$$

Notice now that $\sum_{j=1}^{h} V_\delta(s_{i_j}, \rho_j)$ is equal to the overall volume Ψ of the system, plus $h \leq k$ contributions of size Ψ/k. As a consequence,

$$m_{MC}(x) \leq 2\ln(2k) \sum_{j=1}^{h} V_\delta(s_{i_j}, \rho_j) \leq 4\ln(2k)\Psi,$$

QED and the theorem is proved.

4.2 Between APX and PTAS

AN ASYMPTOTIC approximation scheme is a weaker form of approximation scheme based on the idea that the performance ratio of the returned solution may improve as the optimal measure increases. As we will see in this section, NPO problems exist for which no PTAS can be developed (unless P = NP) but that admit an asymptotic approximation scheme. Let us start with the formal definition of this latter notion.

◀ Definition 4.1
Asymptotic approximation scheme

An NPO *problem* $\mathcal{P}$ *admits an* asymptotic approximation scheme *if there exist an algorithm* $\mathcal{A}$ *and a constant* k *such that, for any instance* x *of* $\mathcal{P}$ *and for any rational* $r \geq 1$, $\mathcal{A}(x,r)$ *returns a solution whose performance ratio is at most* $r + k/m^*(x)$. *Moreover, for any fixed* r, *the running time of* $\mathcal{A}$ *is polynomial.*

The class PTAS$^\infty$ is the set of all NPO problems that admit an asymptotic approximation scheme. Clearly,

$$\text{PTAS} \subseteq \text{PTAS}^\infty \subseteq \text{APX}.$$

Moreover, it is possible to prove that these inclusions are strict if and only if P $\neq$ NP (see Exercise 4.2).

4.2.1 Approximating the edge coloring problem

In this section we consider MINIMUM EDGE COLORING, that is, the problem of coloring the edges of a graph with the minimum number of colors (see Problem 4.2.1).

Minimum Edge Coloring

INSTANCE: Graph $G = (V, E)$.

SOLUTION: A coloring of E, i.e., a partition of E into disjoint sets $E_1, E_2, \ldots, E_k$ such that, for $1 \leq i \leq k$, no two edges in E_i share a common endpoint in G.

MEASURE: The number of colors, i.e., k.

Given a graph G whose maximum degree is Δ, it is easy to see that the optimal value of MINIMUM EDGE COLORING is greater than or equal to Δ. The next result will show that, on the other hand, this value is never greater than $\Delta + 1$. Deciding which one of the two cases holds is an NP-complete problem (see Bibliographical notes) thus implying that MINIMUM EDGE COLORING is not in PTAS (see Exercise 4.9).

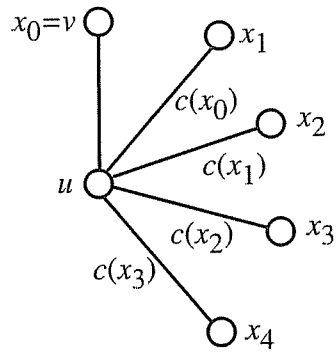

Figure 4.6
The star starting from (u,v)

Theorem 4.6 ▶ *There exists a polynomial-time algorithm $\mathcal{A}$ such that, for any graph G of maximum degree Δ, $\mathcal{A}(G)$ returns an edge-coloring with at most $\Delta+1$ colors.*

PROOF Let $G=(V,E)$ be an instance of MINIMUM EDGE COLORING and let Δ denote the maximum degree of G. The algorithm starts from a new graph $G'=(V,E'=\emptyset)$ (which can be trivially edge-colored with 0 colors) and repeatedly performs the following operations until all edges of E have been *moved* to G':

1. Consider an uncolored edge $(u,v)\notin E'$.
2. Extend the edge-coloring of $G'=(V,E')$ into an edge-coloring of $G''=(V,E'\cup\{(u,v)\})$ with at most $\Delta+1$ colors.
3. Delete (u,v) from E and add it to E'.

We now present step 2 above in detail. Let us assume that a (partial) edge-coloring of G' with at most $\Delta+1$ colors is available such that all edges are colored but edge (u,v). For any node v, in the following we denote by $\mu(v)$ the set of colors that are not used to color edges incident to v. Clearly, if the coloring uses $\Delta+1$ colors, then, for any v, $\mu(v)\neq\emptyset$. In this case, we denote by $c(v)$ one of the colors in $\mu(v)$ (for instance, the color with minimum index).

We can easily find in linear time a sequence $(u,x_0),\ldots,(u,x_s)$ of distinct edges of G' incident to u such that (see Fig. 4.6 where $s=4$):

1. $x_0=v$.
2. For any i with $1\leq i\leq s$, the color of edge (u,x_i) is $c(x_{i-1})$, that is, edge (u,x_i) is colored with a color not used for any edge incident to x_{i-1}.

3. The sequence is maximal, i.e., there is no other edge (u,w) which does not belong to the sequence such that its color is $c(x_s)$.

Note that if $s = 1$, that is, the sequence contains only (u,v), then $\mu(u) \cap \mu(v) \neq \emptyset$: in this case, coloring (u,v) is a trivial task.

We distinguish the following two cases:

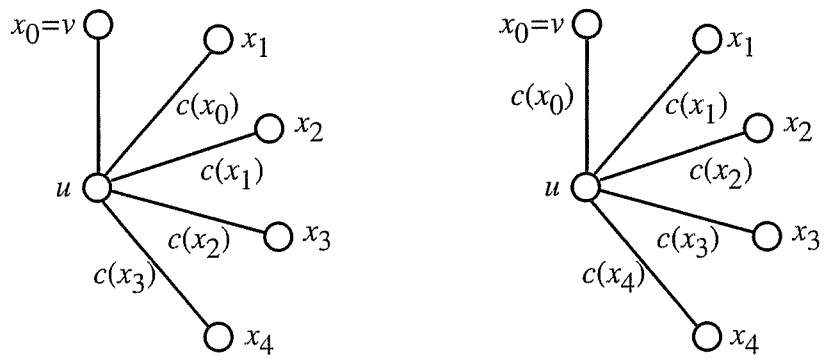

Figure 4.7
The case in which $c(x_4) \in \mu(u)$

1. $c(x_s) \in \mu(u)$: in this case, we can extend the coloring of G' in order to include (u,v) by *shifting* the colors of the sequence. More formally, we start coloring (u,x_s) with $c(x_s)$; then, $c(x_{s-1})$ (that is, the previous color of (u,x_s)) is now in $\mu(u)$, that is, $c(x_{s-1}) \in \mu(u)$. By repeatedly applying this reasoning, we modify the colors of all edges in the sequence but the first one. At the end of this process, we have that $c(v) \in \mu(u)$ so that (u,v) can be colored with $c(v)$ (see Fig. 4.7).

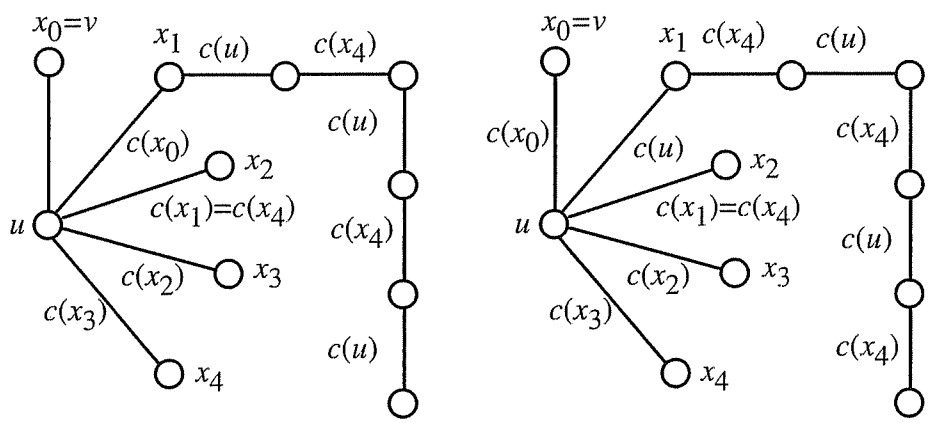

Figure 4.8
The case in which $c(x_4) \notin \mu(u)$ and P_1 does not end in u

2. $c(x_s) \notin \mu(u)$: in this case, one edge (u,x_i) must have been colored with $c(x_s)$ (otherwise, the sequence is not maximal). This implies that $c(x_{i-1}) = c(x_s)$. Let P_{i-1} be the maximal path starting from x_{i-1}

formed by edges whose colors are, alternatively, $c(u)$ and $c(x_s)$ and let w be the last node of this path. Notice that P_{i-1} may have length 0: in such a case, clearly, $w = x_{i-1}$. Two cases may arise.

(a) $w \neq u$: in this case we can interchange colors $c(u)$ and $c(x_s)$ in P_{i-1} and assign color $c(u)$ to (u, x_{i-1}). Note that in this way $c(x_{i-1}) \in \mu(u)$ and, thus, the subsequence preceeding x_{i-1} can be dealed as in Case 1 above (see Fig. 4.8).

(b) $w = u$: in this case, we derive in linear time the path P_s starting from x_s formed by edges whose colors are, alternatively, $c(u)$ and $c(x_s)$. This path cannot intersect P_{i-1}. On the contrary, let z be the first node in P_s that belongs to P_{i-1}. If $z = u$ then there are two edges incident in u with color $c(x_s)$ contradicting the property of a feasible edge coloring of G'. If $z \neq u$, then there must exist three edges incident to z colored with $c(u)$ or $c(x_s)$: once again this contradicts the property of a coloring (see Fig. 4.9).

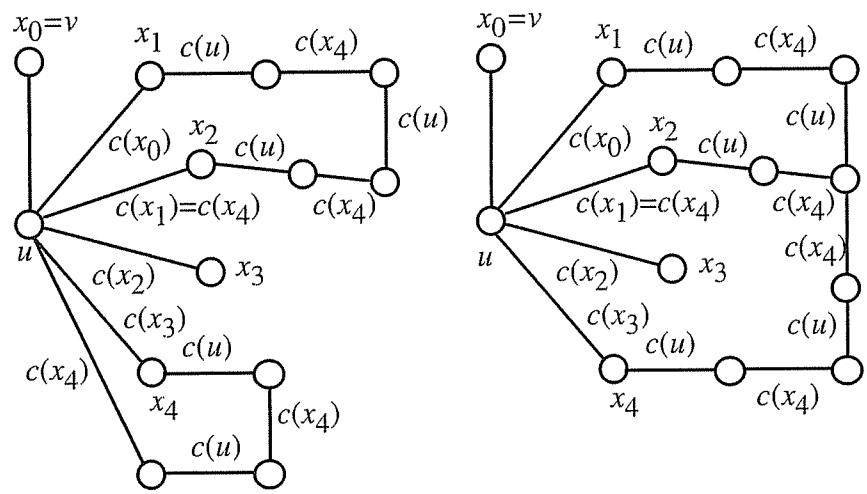

Figure 4.9
P_1 cannot intersect P_4

Since P_s does not intersect P_{i-1}, it cannot end in u: thus, analogously to Case 2.1 above, we can interchange colors $c(u)$ and $c(x_s)$ in P_s and assign color $c(u)$ to (u, x_s). Finally, the subsequence preceeding x_s can be dealed as in Case 1 above (see Fig. 4.10).

In both cases, we have obtained a valid edge-coloring of G' with at most $\Delta + 1$ colors. Since the updating of the coloring of G' has to be done $|E|$ times, it also follows that the running time of the algorithm is bounded by a polynomial.

QED

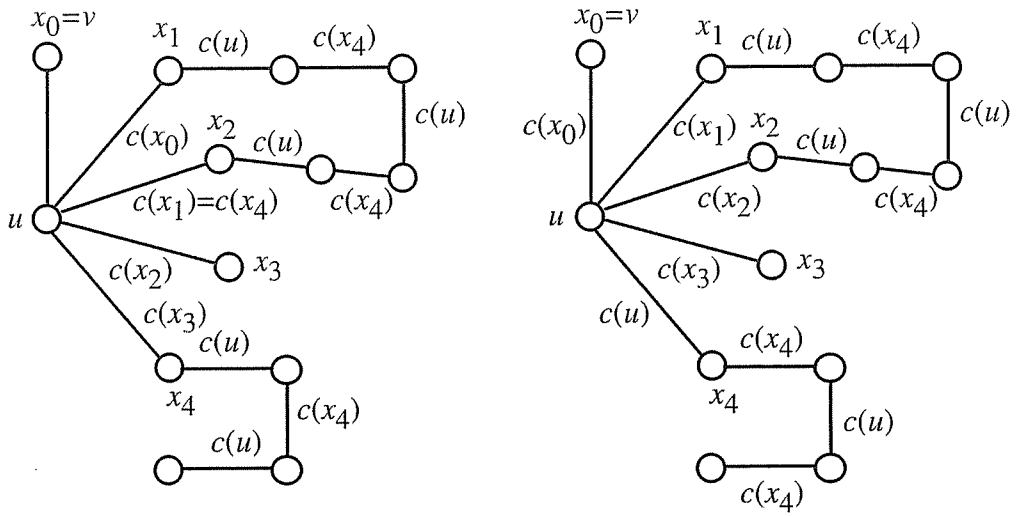

Figure 4.10
The interchange of colors ir P_4 and the shift in the remaining sequence

The above theorem implies that a polynomial-time algorithm exists such that, for any graph G, the performance ratio of the solution returned by the algorithm is at most

$$\frac{m^*(G)+1}{m^*(G)} = 1+\frac{1}{m^*(G)},$$

that is, MINIMUM EDGE COLORING admits an asymptotic approximation scheme with $k = 1$ (observe that a similar argument applies to any NPO problem that admits a polynomial-time algorithm whose absolute error is bounded by a constant). In conclusion, we have the following result.

MINIMUM EDGE COLORING *belongs to the class* PTAS$^\infty$. ◀ Theorem 4.7

4.2.2 Approximating the bin packing problem

In this section we will prove that MINIMUM BIN PACKING belongs to PTAS$^\infty$: note that, in this case, no polynomial-time algorithm whose absolute error is bounded by a constant exists (unless P=NP). We will provide an asymptotic approximation scheme for MINIMUM BIN PACKING, which is based on a combination of the First Fit algorithm and a partitioning technique and is structured into the following five steps.

1. Eliminate *small* items.
2. Group the remaining items into a constant number of size values.
3. Find an optimal solution of the resulting instance.

4. Ungroup the items.
5. Reinsert *small* items.

We will first describe step 3, then steps 1 and 2 along with their "inverse" steps 5 and 4, respectively.

Solving instances of restricted bin packing

For any integer constant $c > 0$ and for any rational constant $\delta \leq 1$, let us consider a restricted version of MINIMUM BIN PACKING in which, for any instance, there are at most c different sizes for the items (i.e., the cardinality of the range of the size function s is bounded by c) and the size of each item is at least $B \cdot \delta$, where B denotes the size of the bins. For the sake of clarity, we will denote this restriction as MINIMUM (c,δ)-RESTRICTED BIN PACKING. Observe that an instance of this problem can be described by the size B of the bins and a multiset $I = \{s_1 : n_1, s_2 : n_2, \ldots, s_c : n_c\}$ in which, the pair $s_i : n_i$, with $1 \leq i \leq c$, denotes the number n_i of items having size s_i.

Example 4.5 ▶ Let us consider the following instance of MINIMUM $(3,3/8)$-RESTRICTED BIN PACKING:

$$I = \{3:4, 5:2, 7:1\} \quad \text{and} \quad B = 8.$$

In this case we have four items with size 3, two with size 5, and one with size 7.

We now show how an instance of MINIMUM (c,δ)-RESTRICTED BIN PACKING can be solved in time $O(n^q)$ where n denotes the number of items and q depends on c and δ only. Equivalently, we show that, for any two constants c and δ, MINIMUM (c,δ)-RESTRICTED BIN PACKING is solvable in polynomial-time.

Let (I,B) be an instance of MINIMUM (c,δ)-RESTRICTED BIN PACKING. The *type* of a bin is a c-vector $\vec{t} = (t_1, \ldots, t_c)$ of integers with $0 \leq t_i \leq n_i$ such that $\sum_{i=1}^{c} t_i s_i \leq B$. In other words, the type specifies for each element s_i of the range of s the number of items having size s_i that are contained in the bin. Observe that for each type $\vec{t}$,

$$\sum_{i=1}^{c} t_i \leq \frac{1}{\delta} \sum_{i=1}^{c} t_i \frac{s_i}{B} \leq \frac{1}{\delta}.$$

This implies that the number of distinct bin types is bounded by the number of ways of choosing c integers whose sum is less than $\lfloor 1/\delta \rfloor$. It is easy to prove that such a number is equal to

$$q = \binom{c + \lfloor \frac{1}{\delta} \rfloor}{\lfloor \frac{1}{\delta} \rfloor}$$

(see Exercise 3.10). Therefore, in any instance of MINIMUM (c,δ)-RESTRICTED BIN PACKING, the number of possible bin types depends on c and δ and does not depend on n.

◀ Example 4.6

Let us consider the instance of Example 4.5. In this case we have that $c = 3$ and $\lfloor 1/\delta \rfloor = 2$. Then

$$q = \binom{5}{2} = 10.$$

Indeed, the possible bin types are only six (i.e., $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, $(1,0,0)$, $(1,1,0)$, and $(2,0,0)$) since the other four bin types (i.e., $(0,0,2)$, $(0,1,1)$, $(0,2,0)$, and $(1,0,1)$) are not feasible because they violate the bound given by the size of the bin.

Since there are at most q different bin types, a feasible solution can now be described by a q-vector $\vec{y} = (y_1,\ldots,y_q)$ where y_i for $i = 1,\ldots,n$ specifies, for each bin type, the number of bins of that type (observe that $0 \leq y_i \leq n$).

◀ Example 4.7

Let us consider again the instance of Example 4.5. A feasible solution can be the one using two bins of type $(2,0,0)$, two bins of type $(0,1,0)$, and one bin of type $(0,0,1)$. One optimal solution, instead, uses two bins of type $(1,1,0)$, one bin of type $(2,0,0)$, and one bin of type $(0,0,1)$.

It is clear that the number of feasible solutions is bounded by $O(n^q)$, which implies that the instance can be solved in time $O(n^q p(n))$ where p is a polynomial by exaustively generating all these feasible solutions (see Exercise 4.10).

Grouping and ungrouping items

Given an instance x of MINIMUM BIN PACKING, let us assume that the n items are ordered according to their size values so that

$$s(u_1) \geq s(u_2) \geq \cdots \geq s(u_n).$$

For any integer $k \leq n$, let $m = \lfloor n/k \rfloor$ and partition the n items into $m+1$ groups G_i with $G_i = \{u_{(i-1)k+1},\ldots,u_{ik}\}$ for $i = 1,\ldots,m$ and $G_{m+1} = \{u_{mk+1},\ldots,u_n\}$.

We then define a new instance x_g of MINIMUM BIN PACKING with the same bin size B that, for $i = 2,3,\ldots,m+1$, contains an item of size $s(u_{(i-1)k+1})$ for each item of instance x that belongs to G_i (that is, the size of all items in the ith group are made equal to that of the largest item in G_i). Note that there at most mk items in x_g.

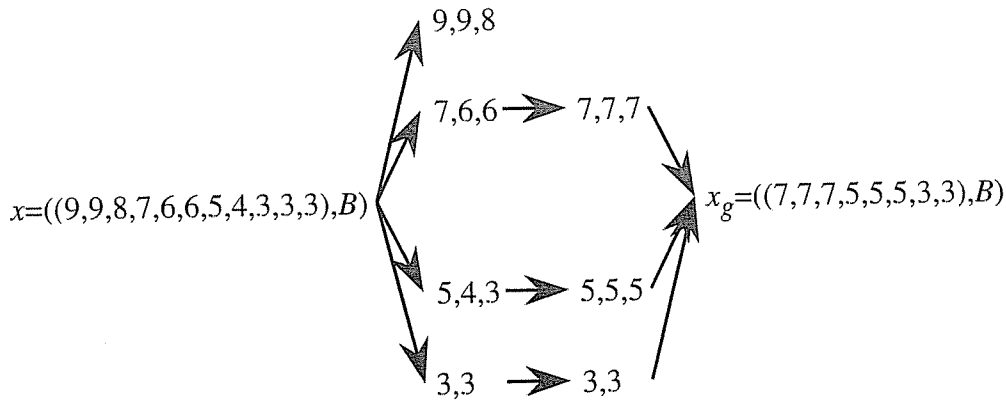

Figure 4.11
An example of grouping items

Example 4.8 ▶ Let us consider the instance x formed by 11 items whose size values are 9, 9, 8, 7, 6, 6, 5, 4, 3, 3, and 3, respectively, and let $k = 3$. We have four groups: $G_1 = \{u_1, u_2, u_3\}$, $G_2 = \{u_4, u_5, u_6\}$, $G_3 = \{u_7, u_8, u_9\}$, and $G_4 = \{u_{10}, u_{11}\}$. The instance x_g contains eight items (see Fig. 4.11): three items of size 7 (corresponding to the items in G_2), three items of size 5 (corresponding to the items in G_3), and two items of size 3 (corresponding to the items in G_4).

Clearly, any packing for the items in x can be transformed into a packing for the items in x_g with the same number of bins by eliminating items in the last group and then substituting each item u in G_i with an item of x_g whose size is equal to $s(u_{ik+1})$ (that is, less than or equal to $s(u)$). Moreover, given a packing of the items in x_g we can obtain a packing of the items in x by simply adding k bins in which we can put the first k items. This implies that

$$m^*(x_g) \leq m^*(x) \leq m^*(x_g) + k, \tag{4.3}$$

that is, if we are able to optimally solve x_g, then we can find a solution for x whose absolute error is at most k.

Dealing with small items

Let x be an instance of bin packing and, for any rational constant $\delta \in (0, 1/2]$, let x_δ be the instance obtained by eliminating all items whose size is less than δB. Given a packing of x_δ with M bins, we can use the First Fit approach to reinsert small items. That is, for each of these items, we insert it into the first bin that can contain it: if it does not fit into any of the currently available bins, then a new bin is created.

At the end of the above procedure two cases are possible.

1. No new bin has been created and the packing uses M bins.

2. $M' \geq 1$ new bins have been created. In this case, similarly to the analysis made in Sect. 2.2.2, we can show that all bins except at most one have an empty space that is at most δB. This implies that

$$(1-\delta)(M+M'-1) \leq \frac{\sum_{i=1}^{n} s(u_i)}{B} \leq m^*(x),$$

that is,

$$M+M' \leq \frac{1}{1-\delta} m^*(x)+1 \leq (1+2\delta)m^*(x)+1.$$

In conclusion, given a packing of x_δ with M bins, we can find in polynomial time a solution for x whose measure is at most

$$\max(M, (1+2\delta)m^*(x)+1). \tag{4.4}$$

We have thus completed the description of the five steps of the proposed algorithm for MINIMUM BIN PACKING which is summarized in Program 4.4. Observe that if $r \geq 2$, the First Fit algorithm achieves the desired performance ratio: hence, the input of the algorithm, without loss of generality, is restricted to values of r smaller than 2. We can finally state the main result of this section.

Program 4.4 is an asymptotic polynomial-time approximation scheme for MINIMUM BIN PACKING. ◀ Theorem 4.8

PROOF

To show that, for any fixed $r < 2$, Program 4.4 runs in polynomial time, it suffices to prove that an optimal solution for $x_{\delta,g}$ can be found in polynomial time. Indeed, $x_{\delta,g}$ is an instance of MINIMUM $(\lfloor n'/k \rfloor, \delta)$-RESTRICTED BIN PACKING: since, for any fixed r, both $\lfloor n'/k \rfloor$ and δ are constant, it follows that an optimal solution for $x_{\delta,g}$ can be computed in time $O(n^q p(n))$ where q depends on r and p is a polynomial.

Let us now compute the performance ratio of the packing obtained by the algorithm. To this aim, first observe that the measure of the solution computed by Program 4.4 is $m^*(x_{\delta,g})+k$. Since all items in x_δ have size at least δB, it follows that $\delta n' \leq m^*(x_\delta)$ and, therefore,

$$k \leq \frac{(r-1)^2}{2} n' + 1 = (r-1)\delta n' + 1 \leq (r-1)m^*(x_\delta)+1.$$

From Eq. 4.3, it follows that the algorithm packs the items of x_δ into at most

$$m^*(x_{\delta,g})+k \leq m^*(x_\delta)+(r-1)m^*(x_\delta)+1 = rm^*(x_\delta)+1.$$

Program 4.4: Asymptotic Bin Packing

input Instance x of MINIMUM BIN PACKING, rational r with $1 < r < 2$;
output Solution whose measure is at most $rm^*(x) + 1$;
begin
  $\delta := (r-1)/2$;
  Let x_δ be the instance obtained by removing items whose size is less than δB;
  $k := \lceil (r-1)^2 n'/2 \rceil$ where n' is the number of items of x_δ;
  Let $x_{\delta,g}$ be the instance obtained by grouping x_δ with respect to k;
  Find an optimal solution for $x_{\delta,g}$ with measure $m^*(x_{\delta,g})$;
  Insert the first k items of x_δ into k new bins;
  Apply First Fit approach to reinsert the small items;
  return packing obtained
end

Finally, by plugging $r = (1+2\delta)$ into Eq. 4.4, we obtain that the total number of used bins is at most

$$\max(rm^*(x_\delta) + 1, rm^*(x) + 1) \leq rm^*(x) + 1,$$

QED which provides the desired performance ratio.

The above theorem states that MINIMUM BIN PACKING belongs to PTAS$^\infty$. It is indeed possible to prove a stronger result that states the existence of an asymptotic approximation scheme whose running time is polynomial both in the length of the input and in $1/(r-1)$ (see Bibliographical notes).

FPTAS^{\infty}

4.3 Exercises

Exercise 4.1 Prove that the analysis of Theorem 4.1 is tight. (Hint: generalize the instance of Example 4.1.)

Exercise 4.2 (*) Prove that if P $\neq$ NP then PTAS $\neq$ PTAS$^\infty$ and PTAS$^\infty \neq$ APX.

Exercise 4.3 Prove that Program 4.5 colors any graph G with n vertices using $O(m^*(G)\log n)$ colors.

Exercise 4.4 Prove that any 2-colorable graph G can be colored in polynomial time using at most 2 colors.

Exercise 4.5 Prove that any graph G of maximum degree $\Delta(G)$ can be colored in polynomial time using at most $\Delta(G)+1$ colors.

Program 4.5: OptIS Graph Coloring

input A graph $G = (V, E)$;
output A coloring of G with i colors;
begin
  $i := 1$;
  while G is not empty **do**
  begin
    find a maximum independent set S_i in G;
    color vertices of S_i with color i;
    delete from G all vertices in S_i;
    $i := i + 1$
  end
end.

Program 4.6: 3-Coloring

input 3-colorable graph $G = (V, E)$;
output Coloring of vertices of G;
begin
  $H := G; n := |V|$;
  $i := 1$;
  while the maximum degree in H is at least $\lceil \sqrt{n} \rceil$ **do**
  begin
    Let v be the vertex of maximum degree in H;
    Let $H_N(v)$ be the graph induced on H by the neighbors of v;
    Color $H_N(v)$ with colors $i, i+1$;
    Color v with color $i+2$;
    $i := i+2$;
    $H :=$ the subgraph of H obtained by deleting v and its neighbors
  end;
  Color all nodes in H with $\Delta(H) + 1$ colors
end.

Exercise 4.6 Prove that, for any 3-colorable graph G, Program 4.6 colors G in polynomial time with at most $3\lceil \sqrt{n} \rceil$ colors. (Hint: use the previous two exercises in order to evaluate the running time.)

Exercise 4.7 (**) Prove that any k-colorable graph G, can be colored in polynomial time with at most $2k\lceil n^{1-1/(k-1)} \rceil$ colors. (Hint: extend the ideas contained in Program 4.6.)

Exercise 4.8 Prove that the two integer linear programming formulations of MINIMUM MULTI-CUT are, indeed, equivalent.

Exercise 4.9 Use the gap technique to show that MINIMUM EDGE COLORING is not in PTAS.

Exercise 4.10 For any fixed c and δ, give a polynomial-time algorithm to solve MINIMUM (c,δ)-RESTRICTED BIN PACKING.

4.4 Bibliographical notes

THE NOTION of input-dependent approximation algorithm is as old as that of constant approximation algorithm: they both appeared in [Johnson, 1974a] which is widely considered as one of the starting points of the systematic study of the complexity of approximation.

The approximation algorithm for MINIMUM SET COVER, which is described in Sect. 4.1.1, was proposed in [Johnson, 1974a] and in [Chvátal, 1979]. This algorithm is optimal: in [Feige, 1996, Raz and Safra, 1997], in fact, it is proved that, for any $\varepsilon > 0$, there is no $(\ln n - \varepsilon)$-approximation algorithm for MINIMUM SET COVER, unless some likely complexity theoretic conjectures fail. It is worth pointing out that it took more than twenty years to close this gap and that MINIMUM SET COVER is one of the very few problems for which optimal approximation algorithms have been proved.

The $O(n/\log n)$-approximate algorithm for MINIMUM GRAPH COLORING is due to [Johnson, 1974a] (from this reference, Exercise 4.3 is also taken): this algorithm is not optimal. Indeed, the best known approximation algorithm for MINIMUM GRAPH COLORING is due to [Halldórsson, 1993a] and has a performance ratio $O(n(\log\log n)^2/(\log n)^3)$. On the other hand, in [Bellare, Goldreich, and Sudan, 1998] it is shown that this problem is not approximable within $n^{1/7-\varepsilon}$ for any $\varepsilon > 0$ unless P = NP. Exercises 4.6 and 4.7 are due to [Wigderson, 1983].

The max-flow min-cut theorem in single-commodity networks was introduced in [Ford and Fulkerson, 1956]. The first corresponding approximation result for the multi-commodity case has been presented in [Leighton and Rao, 1988]. In this paper, the authors consider the related SPARSEST CUT problem, where a cut E' is required on a weighted graph $G = (V,E,c)$ on which k pairs $\{s_i,t_i\} \in V^2$ are defined, such that the ratio $\rho(E') = \sum_{e\in E'} c(e)/\sum_{i\in I(E')} d_i$ is minimum, where $i \in I(E')$ if and only if s_i and t_i are disconnected by the removal of E'. In the same paper an $O(\log n)$ approximation algorithm for this problem is given and a relationship between this problem and the BALANCED CUT is shown, where BALANCED CUT requires, given a value $\alpha \in (0,1)$, to find a cut of minimum cost which disconnects the graph into two subgraphs each of size at least αn. Such a

problem presents particular relevance as a building block for the design of divide and conquer algorithms on graphs.

The algorithm presented in Sect. 4.1.3 for MINIMUM MULTI-CUT is from [Garg, Vazirani, and Yannakakis, 1996] and has been proved to be applicable to the approximate solution of SPARSEST CUT (and of BALANCED CUT) in [Kahale, 1993]. Other approximation results for SPARSEST CUT has been derived in [Klein, Rao, Agrawal, Ravi, 1995, Plotkin, Tardos, 1995, Linial, London, Rabinovich, 1995, Aumann and Rabani, 1995].

The definition of an asymptotic approximation scheme is taken from [Motwani, 1992] and is based on the notion of asymptotic performance ratio as given in [Garey and Johnson, 1979].

Theorem 4.6 is a fundamental result of graph theory and appeared in [Vizing, 1964]: observe that this theorem also implies that MINIMUM EDGE COLORING is 4/3-approximable. Deciding whether a graph is edge-colorable with Δ colors was shown to be NP-complete in [Holyer, 1981].

The asymptotic approximation scheme for MINIMUM BIN PACKING appeared in [Fernandez de la Vega and Lueker, 1981]: our presentation follows that of [Motwani, 1992]. Indeed, this scheme has been improved in [Karmarkar and Karp, 1982]: in this paper, an asymptotic fully polynomial-time approximation scheme is obtained by means of mathematical programming relaxation techniques and of the ellipsoid method introduced in [Khachian, 1979].